

Szegedi Tudományegyetem
Informatikai Intézet

SZAKDOLGOZAT

Ocztos Károly Levente
2024

Szegedi Tudományegyetem
Informatikai Intézet

Magyarországi Madarak mobilalkalmazás

Szakdolgozat

Készítette:

Ocztos Károly Levente
programtervező
informatikus szakos
hallgató

Témavezető:

Dr. Bilicki Vilmos
egyetemi adjunktus

Szeged
2024

Feladatkiírás

A szakdolgozat keretein belül a cél egy olyan mobilalkalmazás elkészítése, ami Magyarországon előforduló madarokról nyújt információkat, és a felhasználók számára madárhatározó funkciót kínál. Emellett alkalmas megfigyelések létrehozására és szerkesztésére.

Az alkalmazásban a felhasználók madarakat kereshetnek, amelyekről részletes információkat tekinthetnek meg, valamint madárhatározóval lehetőség van előre meghatározott jellemzők alapján keresni madarakat.

Az alkalmazás a felhasználók számára offline módot is biztosít, hogy a felhasználók akkor is információkhoz juthassanak a madarokról, ha nem csatlakoznak az internethez vagy éppen nincs térerő.

A fajok megfigyelései közvetlenül rögzíthetők az alkalmazásban, ezek szerkesztésére van lehetőség. Felhasználói fiókot is létre lehet hozni, ami segítségével a felhőben tárolhatja el a felhasználó a megfigyeléseit.

Az alkalmazás grafikus felhasználói felülete könnyű kezelhetőséget biztosít, és a reszponzív oldalak széleskörű eszközökön való elérhetőséget tesznek lehetővé.

Tartalmi összefoglaló

A téma megnevezése

Egy android mobilalkalmazás, ami a magyarországi madarokról nyújt információkat és madárhatározó funkciókkal rendelkezik.

A megadott feladat megfogalmazása

Az alkalmazás madarakat listáz, mely listát kereséssel, vagy madárhatározóval lehet szűkíteni. Megfigyelések létrehozására is lehetőséget nyújt, melyeket módosítani, törölni lehet. Az alkalmazás célja megismertetni a madarakat és segítséget nyújtani a beazonosításukhoz.

A megoldási mód

A Magyarországi Madarak mobilalkalmazás specifikációja után megterveztem az alkalmazást. Ez a fázis magába foglalja a piackutatást és a használt technológiák alaposabb megismerését, diagramok elkészítését. A fejlesztést xml és natív java nyelveken valósítottam meg, az adatok tárolásához pedig Google Firebase és RoomDatabase szolgáltatásokat használtam. A folyamatot az alkalmazás ellenőrzésével és tesztelésével zártam le.

Alkalmazott eszközök, módszerek

A diagramokat a draw.io weboldalon készítettem. Az Android Studio fejlesztői környezetben fejlesztettem az android mobilalkalmazást. A megjelenítésért felelős részeket XML, a mögöttes logikát natív Java nyelven programoztam le. A GitHub platformot használtam egyaránt verziókövetéshez és a fejlesztés feladatainak kezeléséhez. A helyi adatokat RoomDatabase, az online adatokat Google Firebase segítségével tároltam.

A fejlesztési folyamathoz agilis fejlesztési modellt választottam Scrum-szerű módszertannal: miután megterveztem az alkalmazást, úgy elkezdtem a fejlesztését. A fejlesztés során az igényekhez és a szoros határidőkhöz igazítottam a terveket. A módszertan lehetővé tette, hogy a folyamat során felfedezett technikákat és technológiákat könnyedén felhasználhassam a végeredmény kialakításához.

Elért eredmények

A kezdeti tervek során kitűzött célokat sikeresen megvalósítottam, a fejlesztés során kialakult új célokkal együtt. Emellett mélyebb tudást szereztem az elkészítés során használt eszközökről.

Kulcsszavak

Android Studio, Google Firebase, adatkezelés

Tartalomjegyzék

TODO: tartalomjegyzék

MOTIVÁCIÓ

Manapság egyre több ember dönt úgy, hogy túrázik a természetben, ami remek lehetőség a kikapcsolódásra és a friss levegőn való feltöltődésre. Egy-egy kirándulás vagy séta során gyakran találkozhatunk érdekes madarakkal. Van, hogy eltűnődünk azon, hogy milyen madár lehet, milyen különleges tulajdonságai vannak, vagy hogy honnan érkezett. Az ilyen pillanatok azonban gyakran elszállnak, mert nem mindig áll rendelkezésünkre a megfelelő eszköz vagy tudás, hogy választ kapjunk a kérdéseinkre.

A Magyarországi Madarak mobilalkalmazás pontosan ezt a célt szolgálja. Ez az alkalmazás lehetővé teszi, hogy bárhol és bármikor információt szerezzünk a minket körülvevő madarokról. Legyen szó egy hétvégi túráról, egy délutáni sétáról a parkban, vagy akár arról, hogy az otthonunk közelében látunk meg egy különleges madarat – az applikáció segítségével azonosíthatjuk, és bővebb információkat tudhatunk meg róla.

Azonban az alkalmazás nemcsak az azonosítást könnyíti meg, hanem lehetőséget ad arra is, hogy saját megfigyeléseinket rögzítsük. Így személyes madárfigyelési naplót vezethetünk, amelyben visszanézhetjük, hogy milyen madarakkal találkoztunk az utazásaink során. Ez nemcsak hasznos és szórakoztató, hanem segíthet jobban megérteni a természet világát és közelebb hoz minket a környezetünkhöz.

1. Területi áttekintés

A piackutatás során a célom olyan alkalmazások megtalálása volt, amelyek madárhatározó szerepet töltenek be. Az elsődleges szempont az volt, hogy magyarországi madarakról tároljon információkat és mobilalkalmazás legyen, de a magyar nyelvűség is kiemelkedő szerepet játszott, mivel elsősorban magyaroknak szánt alkalmazás elkészítése volt a cél.

Az alkalmazások keresése közben azt vettem észre, hogy nem áll túl sok ilyen alkalmazás rendelkezésünkre, magyarok számára többnyire automatikusan fordított, félmagyar mondatokkal tarkítottak elérhetőek. Az alkalmazások hiánya miatt a szempontok kiválasztása főként az általam elvárt funkciók voltak nagyrészt, és ezek egészültek ki a kutatás során talált egyéb funkciókkal.

A kutatás során felmért alkalmazások teljes listáját és funkcióit az 1.1. ábra mutatja be. Ezeknek az alkalmazásoknak köszönhetően sikerült felmérnem, hogy a Magyarországi Madarak mobilalkalmazásnak milyen felhasználói igényeket kell kielégítenie. Ötleteket adtak és példát mutattak a tervezett funkciók megvalósításához. Arra is rávilágítottak, hogy milyen hiányosságokat tudok orvosolni a fejlesztés során, és teljesebb képet kaptam az elérendő céljaimhoz.

	Magyar nyelv	Térség	Lexikon	Madárhatározó	Megfigyelés rögzítés	Képfelismerő	Hangfelismerő
Madárhatározó (MME+FIE)	igen	csak MO	igen	igen	nem	nem	nem
Merlin Bird ID	részben	világ (földrész szintű)	igen	igen	igen	igen	igen
EBird	részben	világ (ország szintű)	nem	nem	igen	nem	nem

1.1. ábra – Piackutatás

1.1. Madárhatározó

A Madárhatározó Magyarország első számú madárhatározó alkalmazása, a Magyar Madártani és Természetvédelmi Egyesület (MME) és a Farkaskölykök Ifjúsági Egyesület közös munkája. Ezt a mobilalkalmazás vettem alapul a Magyarországi Madarak mobilalkalmazás megtervezése közben, hiszen ez az egyetlen, ami kifejezetten magyar nyelven készült.

Az alkalmazásban található egy „Ez mi lehet?” modul, ami madárhatározó jelleggel működik. Itt kiválasztható a határozni kívánt madár alakja, színei és élőhelye, és ezek alapján jelenít meg találatokat. Rendelkezik egy „Lexikon” modullal is, ahol az alkalmazás és egyben Magyarországon előforduló összes madár megtalálható. Egy madarat kiválasztva megnyílik

egy részletező oldal, ahol sok érdekes információt tudhatunk meg a madárról. Maga az alkalmazás rendelkezik egy „Játék” modullal is, ami a madárfelismerő tudásunkat teszteli.

1.2. Merlin Bird ID

A Merlin Bird ID mobilalkalmazás a Cornell Lab of Ornithology világhírű kutatóintézet, amely a madarak és más vadon élő állatok tanulmányozásával foglalkozik. Az ő mobilalkalmazásuk igen fejlett, azonban alapvetően angol nyelvű, magyar csupán gépi fordítással érhető el és csak részlegesen.

Rendelkezik „Step by Step”, lépésről lépésre, madárhatározóval, ami pár előzetes szempont alapján jelenít meg találatokat. Ilyen szempontok a madár tollainak színei, mérete, milyen időszakban volt megfigyelve a madár és hol, vagy éppen mit csinált. A rengeteg nemzetközi adatuknak és méréseiknek köszönhetően rendelkezik az alkalmazás hangfelismerővel is, ami a madár hangjáról azonosítja be a fajt. Található még benne egy képfelismerő is, de a sok zavaró tényező miatt sok esetben nem tudja megfelelően felismerni a megfigyelt madarat. Lehetőségünk van madárhatározás után elmenteni a találkozásunkat, így megörökíthetjük az eseményt.

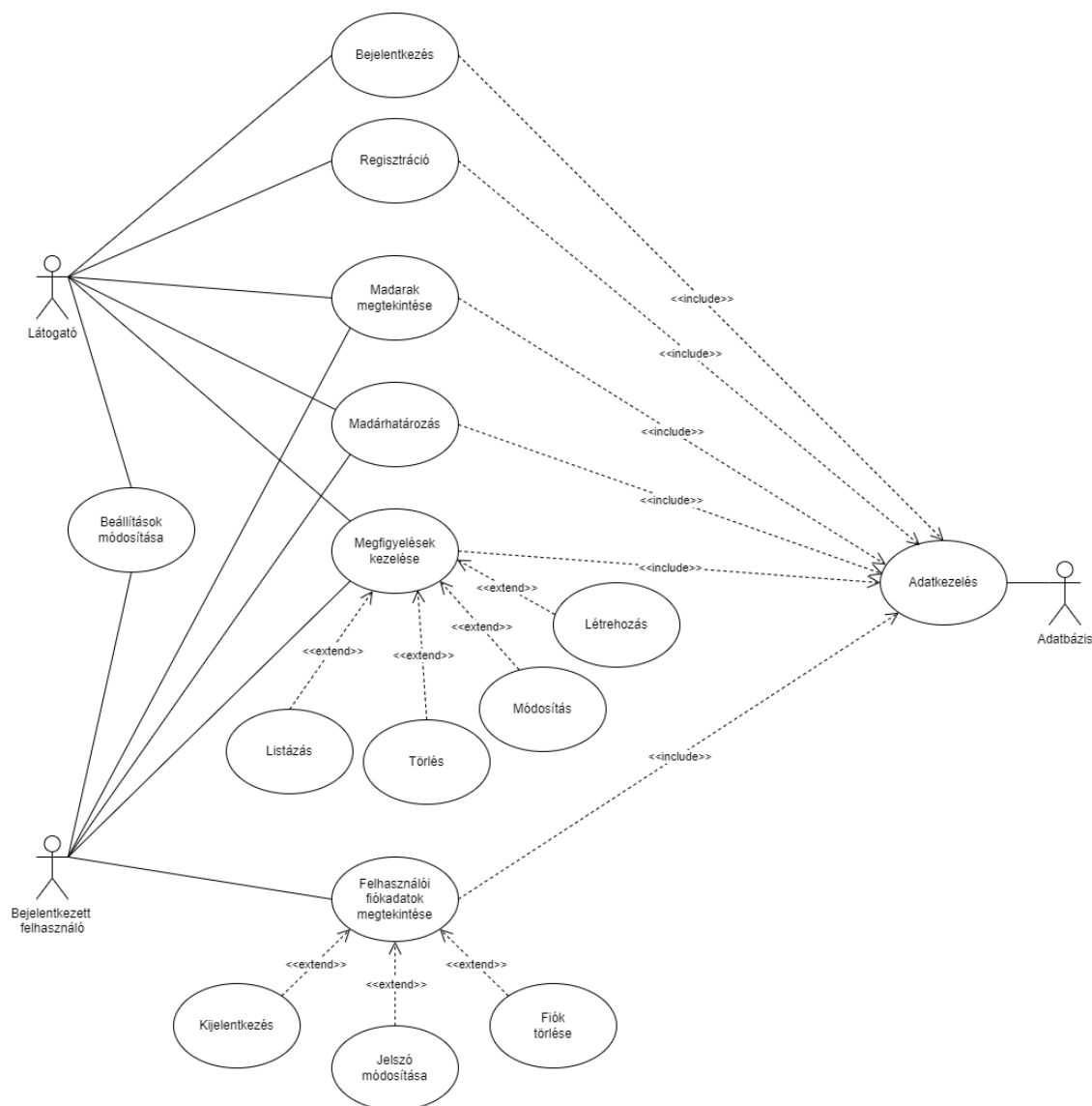
1.3. eBird

Az eBird mobilalkalmazása egy globális megfigyelésrögzítő alkalmazás. Ami szintén a Cornell Lab of Ornithology által készült, és így a Merlin Bird ID-val szorosan együttműködhet.

Az alkalmazás leginkább megfigyelések rögzítésére szolgál, mintsem tájékozódásra a madarokról. Rendelkezik egy hatalmas globális adatbázissal, ami menti a megfigyeléseket. Idő és hely megadásával, majd a megfigyelt madár kiválasztásával elmenthető a madár megfigyelése és ezzel hozzájárulhatunk a madarak szokásainak megfigyeléséhez és elemzéséhez. Amennyiben szeretnénk, a Merlin Bird ID-val készített megfigyelésünket meg tudjuk osztani az eBird-del is, ahol így bekerül a globális adatbázisba is. Az eBird alkalmazásban lehetőségünk van arra is, hogy megtekintsük a közelben jelentett megfigyeléseket, így ha szerencsések vagyunk, mi magunk is megláthatjuk az adott faj egyedeit.

2. Funkcionális specifikáció

A 2.1. ábrán látható az alkalmazás Use Case (Használati eset) diagramja. A felhasználók két csoportba tartoznak: látogató és bejelentkezett felhasználó. Ezek a csoportok többnyire ugyanazokat a funkciókat érik el, különbözőségük mögöttes üzleti logika szintű.



2.1. ábra – Use Case diagram

Ahogy az a diagramon is látható, a Látogatók számára elérhető minden funkció a Felhasználói adatok megtekintésén kívül, viszont a megfigyelések előzetes regisztráció és bejelentkezés nélkül csupán helyileg tárolódnak el.

A bejelentkezett felhasználók hozzáférnek a saját fiókjukhoz. Emellett az ő megfigyeléseik már a felhőben is eltárolódnak, nem csak az eszközön, így más eszközökre is magunkkal vihetjük adatainkat.

2.1. Autentikáció

Az alkalmazás használatához nem kötelező bejelentkezni, és enélkül is elérhető az összes funkció. Azonban, ha szeretnénk, hogy megfigyeléseink ne vesszenek el, bejelentkezés szükséges. Elsődlegesen a beállítások menüpontból érhető el a bejelentkezés és regisztráció felülete. Az autentikálást a Google Firebase beépített szolgáltatásával valósítottam meg.

2.1.1. Regisztráció

Fiók létrehozása két módon is lehetséges. Az egyik egy alap e-mail cím és jelszó, valamint megerősítő jelszó megadásával történik. A jelszónak legalább 12 karakter hosszúnak kell lennie. E-mail cím esetében ellenőrzésre kerül a megfelelő formátum, és a regisztrálás megpróbálása után a felhasználó arról is tájékoztatást kap, ha a megadott cím már foglalt. Sikeres regisztrációt követően a felhasználó jelzést kap arról, hogy erősítse meg e-mail címét a kiküldött e-mail-ben.

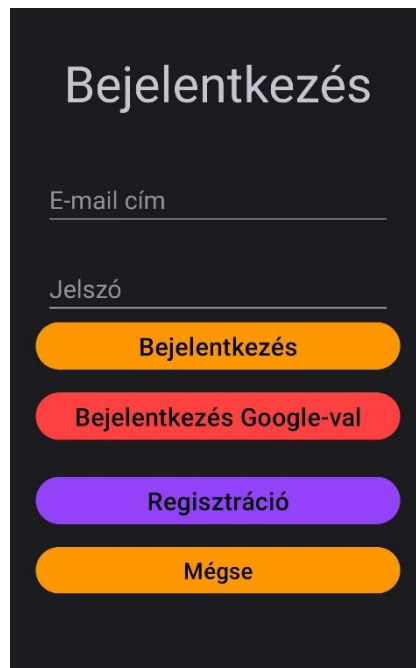
A másik módszer során lehetőség van regisztrálni Google fiókkal. A hitelességről a Google gondoskodik. Amennyiben tudjuk igazolni magunkat a Google-nél, úgy a regisztráció jelszó nélkül és e-mail megerősítése nélkül végbe megy.

2.1.2. Bejelentkezés

A bejelentkezéshez előzetes regisztráció és az e-mail cím visszaigazolása szükséges, ha valamelyik hiányzik, vagy a megadott jelszó helytelen, arról egy hibaüzenet jelenik meg. Google fiókkal való bejelentkezés során ugyanaz az eljárás történik, mint regisztrációnál.

Az alkalmazás mindaddig bejelentkezve marad, amíg ki nem jelentkezünk, így ezt a folyamatot nem kell minden alkalommal megismételni.

Bejelentkezés után az alkalmazás a főoldalra navigál, megváltozik a fejléc, és újabb funkciók válnak elérhetővé.



Bejelentkezés

E-mail cím

Jelszó

Bejelentkezés

Bejelentkezés Google-val

Regisztráció

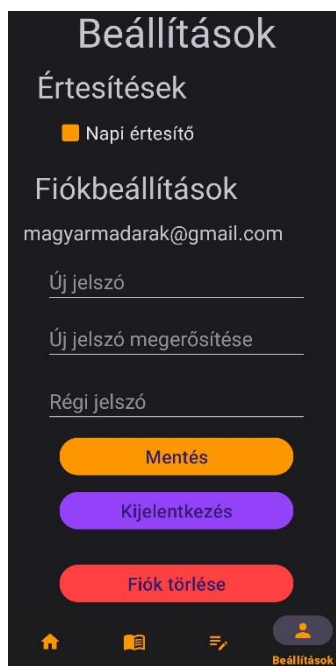
Mégse

2.1.2.1. ábra – bejelentkezési felület

2.1.3. Kijelentkezés

Kijelentkezni a felhasználói fiókunkból minden esetben a Beállításokban tehetjük meg. Itt mutatja a fiókot, amelyikbe be vagyunk jelentkezve. A gombra kattintva automatikusan, azonnal kijelentkezünk és megszűnik az autentikáciánk. A fiókhoz ezután nem férünk hozzá, csak akkor, amikor újra bejelentkezünk. Az oldal automatikusan frissül és tájékoztatja a felhasználót a művelet sikerességéről. A megfigyelések az eszközön maradnak, mivel az az eszközhöz tartoznak, de erről részletesebben a 2.5. Megfigyelések pont alatt lehet olvasni. Ha nem vagyunk bejelentkezve, a kijelentkezés opció nem látszik.

2.2. Beállítások



2.2.1. ábra – beállítások felület bejelentkezett felhasználónak

2.2.1. Értesítések

Beállításokban lehet kezelni az értesítéseket. Ha nem szeretnénk kapni üzeneteket az alkalmazástól, akkor nem kell letiltani, hanem itt ki lehet kapcsolni. A változtatás mentése automatikusan végbemegy, amiről tájékoztatást is ad az alkalmazás.

2.2.2. Fiókbeállítások

A fiókbeállításokban találhatóak a bejelentkezéshez és regisztrációhoz vezető gombok is, amik átvisznek a megfelelő oldalakra.

Bejelentkezett felhasználóknak lehetőségük van a saját profiljuk megtekintésére is. Ezen az oldalon megjelenik az e-mail címük, és itt lehet kijelentkezni vagy akár törölni a felhasználói fiókot. Egyszerű e-mail és jelszavas fiók esetében a fiók törléséhez meg kell adni a jelszót. Jelszó módosításához meg kell adni az új jelszót, valamint meg is kell erősíteni azt, emellett a régi jelszót is igényli a folyamat. A jelszó ellenőrzése itt sem marad el, legalább 12 karakteresnek kell lennie, valamint a jelszónak és megerősítő jelszónak egyezniük kell.

A felhasználó fiókjának törléséhez itt is szükséges megadni a felhasználó jelszavát. A törlés automatikusan végbemegy, sikerességéről tájékoztatja a felhasználót. Az online tárolt adatok, amik a felhasználóhoz köthetők, törlődnek kivétel nélkül, így a megfigyelések is. Azonban a megfigyeléseink elérhetőek maradnak az eszközünkön, mivel csupán a felhőszinkronizálás szűnt meg. Erről bővebben a 2.5. Megfigyelések pont alatt lehet olvasni.

2.3. Madarak megtekintése

A Tudásbázis oldalon kerül listázásra az alkalmazásban található összes madárfaj alfabetikus sorrendben. A listában keresésre is van lehetőség. Ez a madarak nevére szűr, és nem érzékeny a kis- vagy nagybetűkre.

Ha a felhasználó egy listaelemre kattint, azzal megnyílik a madár részletes oldala, ahol a madár összes adata megtekinthető, mint a mérete, természetvédelmi értéke összefoglaló leírása, és még sok más.

TODO: Kép beszúrása

2.3.1. ábra – tudásbázis oldal

2.4. Madárhatározó

Ez a modul azt a célt szolgálja, hogy egy látott madár jellemzőinek megadásával ki tudja találni a felhasználó, milyen faj lehetett a megjelenített találatok alapján. A megadható szempontok között vannak a madár színei, alakja, esetleg élőhelye. Az oldalak között gombok segítségével lehet lapozni előre és vissza. Az utolsó oldalra érve elénk tárul az eredmények oldal, ahol a találatok vannak listázva.

2.4.1. Jellemzők megadása

A jellemzők megadása három fő lépésből, ezáltal három oldalból tevődik össze. Ezek a madár tollainak színei, alakjai és élőhelyei. A lapok között következő és előző gombok segítségével tudunk váltani.

Egy-egy oldalon az összes elérhető jellemző fel van sorolva az adott kategóriában. Ezeket ki tudjuk választani egyesével, de akár több megjelölése is lehetséges egyszerre. A válaszainkat a rendszer automatikusan megjegyzi, ha később visszatérünk, akkor a korábban bejelölt szempontjaink továbbra is ki lesznek választva.

Első lapon a madarak színei vannak felsorolva, az alkalmazásban található összes madár minden színe össze lett gyűjtve egy halmazba. Ezáltal bármelyik szín kiválasztásával érintünk valamilyen madarat. Ugyanígy készült a második oldal, ahol a madarakat jellemző madáralakok vannak felsorolva. Mivel egy fajt van, hogy kettő, három alak is jellemez, így több alak megadását is engedélyezzük. Az összes madár összes alakja ki lett gyűjtve egy halmazba. A módszer a harmadik oldalon sem változott, a madaraknak az élőhelyei kiválasztásánál. Az összes madár minden élőhelye össze lett gyűjtve egy halmazba, majd meg lett jelenítve a felhasználó számára.

2.4.2. Jellemzők megadása

Az utolsó oldalon található az eredmények. A megjelenítésnél figyelembe vannak véve a megadott jellemzők. A találatok között csak olyan fajok szerepelnek, amik a megadott jellemzők mindegyikével rendelkeznek. A megadottakon felül is lehet még jellemzője a madárnak, de a megadottakkal mindenképpen rendelkeznie kell. A végeredmény egy tudásbázishoz hasonló oldal a listázást és felépítést tekintve. A madarak részletező oldala itt is megnyitható.

2.5. Megfigyelések

Ez a modul azt a célt szolgálja, hogy a felhasználók rögzíteni tudják megfigyeléseiket, követni tudják a látott madarakat és rögzítsék élményeiket. Meglévő megfigyeléseikhez könnyen hozzáférhetnek, módosítani, vagy akár törölni is tudják. Felhasználói fiók regisztrálásával a felhőbe is elmenthetik feljegyzéseiket. Az oldal felső részén státuszüzenet van, és ez hogyan befolyásolja a mentéseket.

TODO: Kép beszúrása

2.5.1. ábra – megfigyelések listázó oldala

2.5.1. Megfigyelés listázó oldal

Az oldalra navigálva listázódnak az eszközön található megfigyelések. A listanézetben látható a létrehozás dátuma tájékoztató jelleggel, valamint a megfigyelésnek adott név.

Az elemek sorrendjét a legutolsó módosítás dátuma határozza meg., tehát a legutoljára módosított elem fog szerepelni legfelül.

2.5.2. Megfigyelés létrehozása

Az oldalon található egy Plusz ikon, amire kattintva megnyílik egy megfigyelés létrehozó ablak.

Itt ki kell tölteni a cím mezőt, ahova a megfigyelésünk nevét kell beírni, amivel könnyen be tudjuk azonosítani, miről szól a megfigyelésünk. A megfigyelés időpontját a naptáras választó rendszer és az óra időválasztó segítségével tehetjük meg. Legalul található a megfigyelés leírására szolgáló mező. Bár a kitöltése opcionális, érdemes leírni az élményünket, vagy a megfigyelés körülményeit és a madarat.

Végül van egy törlés gomb, amivel el tudjuk vetni a létrehozást, valamint egy mentés gomb, amivel létre tudjuk hozni a megfigyelésünket. A létrehozás dátumát és legutolsó módosítás dátumát a rendszer automatikusan a mentés pillanatára állítja. Mentést követően bezárja az ablakot és visszavisz a megfigyelések listázó oldalára. A listát automatikusan frissíti.

2.5.3. Megfigyelés szerkesztése

Meglévő megfigyelésekre kattintva megnyílik a szerkesztő oldal, hasonló felépítésben, mint a létrehozó oldal, annyi különbséggel, hogy a mezők ki vannak töltve.

Itt tudjuk módosítani a megfigyelésnek adott elnevezést, a címet, lehet módosítani még a megfigyelés időpontját is és leírását is.

Alul itt is megtalálható a törlés gomb. Erre kattintva azonnal törli a megfigyelést, aminek a sikerességéről értesíti a felhasználót. A mentés gombra kattintva módosítja a módosított értékekre a megfigyelést. A legutolsó módosítás dátumát a mentés gombra kattintáskori időpontra állítja be, és így a megfigyelések lista tetejére kerül. Mindkét gomb esetében visszavisz a megfigyeléseket listázó oldalra. A listát automatikusan frissíti.

2.5.4. Megfigyelés mentési rendszer

Regisztrált és bejelentkezett felhasználóknak a megfigyeléseit automatikusan menti a rendszer a felhőbe.

Bejelentkezést követően minden olyan megfigyelést, ami az eszközön található a bejelentkezett felhasználóhoz társít, és menti a felhőbe azokat. Minden olyan megfigyelést, ami más felhasználóhoz tartozik töröl az eszközről. Ezekhez nincsen hozzáférésünk. Ezt követően minden felhőben található megfigyelést, ami a felhasználóhoz köthető letölt, ha nem található meg az eszközön.

Kijelentkezéskor a megfigyelések ugyanúgy megtalálhatóak lesznek az eszközön, hisz a felhőbe mentés csupán kiegészítő szolgáltatás az adatok mentésére.

Előfordulhat olyan eset, amikor törölünk olyan megfigyeléseket az eszközről, amikre nincsen szükségünk. Azonban azokat, ha már mentettük a felhőbe, akkor onnan nem törölődnek, hisz nincs jogosultságunk hozzá. Ismételt belépéskor letölti, mert ott megtalálhatók.

2.5.5. Felső státuszjelző üzenet

Ez a felső üzenet három állapotot különböztet meg, amik nagy hatással vannak a mentési formára. Üzenet jellegétől függően van, hogy kattintható az elem, amely tulajdonságot a szöveg magában is tartalmaz.

Az egyik szöveg a „*Nem vagy bejelentkezve, a megfigyeléseid nem mentődnek! Kattints ide a bejelentkezéshez!*”. Ez az alap állapot, amikor a felhasználó megfigyelései csak az eszközön elérhetők, a felhőben tárolt adatokhoz nem fér hozzá, az ottani módosításukhoz be kell jelentkezni. Ahogy azt az üzenet szövege is mutatja, a szöveg kattintható, amely hatására megnyílik a bejelentkező ablak. Sikeres bejelentkezés után a megfigyelések oldalára visz

vissza, ahol a szöveg megváltozik automatikusan.

Van egy bejelentkezett és autentikált státuszt jelző szöveg is, ami a „*Be vagy jelentkezve (felhasználó e-mail címe).*”. A felhasználó e-mail címe helyén megjelenik az e-mail címe a bejelentkezett felhasználónak. Ennél az üzenetnél a felhasználó adatai mentődnek a felhőbe, szinkronizálva vannak az ottani adatokkal az eszközön lévők. Ez minden látható megfigyelésre vonatkozik. Ekkor a módosításaink azonnal bekerülnek a felhőbe is.

Az utolsó üzenet a „*Jelenleg offline vagy. Csatlakozz hálózathoz!*”. Ez azt jelzi, hogy nem érhető el internet, így a módosításaink nem mentődhetnek a felhőbe. Ez olyankor is megjelenik, amikor nem vagyunk bejelentkezve, mivel ilyenkor nem elérhető a bejelentkezés sem és erre akarja felhívni a figyelmünket.

3. Felhasznált technológiák

3.1. Natív XML

Az XML egy mobilalkalmazás-fejlesztésben elterjedt nyelv, amely a felhasználói felületek leírására szolgál. Elsődleges célja, hogy strukturált, könnyen olvasható formában határozza meg az alkalmazás vizuális elemeit, valamint azok elrendezését. Az Android fejlesztés során az XML a felhasználói felület kialakításának alapvető eszköze, amely minden vizuális komponenset, például gombokat, szövegeket vagy listákat tartalmaz. Stílusok testreszabására, valamint egységes stílusminták kialakítására is alkalmas.

Az XML fájlok kizárólag a felhasználói felület statikus elemeinek definiálására szolgálnak, így a működés leírására egy másik nyelvet is igénybe kell venni. Ez a tagoltság segít jobban elkülöníteni a részeket, így karbantarthatóbb lesz a kód és szervezettebb, valamint a hibák detektálásában is sokat segít.

Az XML-t használva a vizuális elemek újra felhasználhatóak és testreszabhatóak. Például az alkalmazásban használt stílusokat és szöveges erőforrásokat külön fájlokban lehet tárolni, ami elősegíti a rugalmasságot és a következetességet, elkerülve a kód duplikációt.

Az Android fejlesztés során az XML fájlok moduláris szervezése lehetővé teszi, hogy az egyes képernyők elrendezései külön fájlokban helyezkedjenek el, ezáltal a projektek könnyebben skálázhatóak.

Fejlesztés során natív XML fájlokat használtam, amelyek segítségével részletesen meghatároztam az alkalmazás vizuális elemeit és azok elrendezését, sablonokat készítettem a hasonló elemekhez.

3.2. Natív Java

A Java egy platformfüggetlen, objektumorientált programozási nyelv, ami a mobilalkalmazás-fejlesztés egyik alapeleme. Elsődleges célja, hogy robusztus, biztonságos és skálázható alkalmazásokat hozzon létre, melyek különböző eszközökön és platformokon egyaránt működnek. Az Android fejlesztés során a Java a funkcionalitás, a működés leírásának biztosítására szolgál, legyen szó adatkezelésről, felhasználói interakciók kezeléséről vagy hálózati műveletekről.

A Java nyelv struktúrája lehetővé teszi a könnyen érthető és karbantartható kód írását. Az osztályokra és objektumokra épülő felépítés támogatja az újra felhasználhatóságot és a moduláris tervezést. Az Android alkalmazásokban a Java segítségével valósíthatóak meg a különböző tevékenységek és komponensek működése, például gombokra való kattintás kezelése, kitöltött szövegbeviteli mezők tartalmának feldolgozása vagy adatok lekérdezése

egy adatbázisból.

A Java kiterjedt könyvtárai és keretrendszerei támogatják a különféle fejlesztési igényeket, például a hálózati kapcsolatok, az adatok feldolgozása vagy a felhasználói értesítések kezelése terén. Az Android fejlesztéshez biztosított API-k segítségével a Java lehetővé teszi az alkalmazások eszközspecifikus funkcióinak használatát. A Java nyelv stabilitása és széles körű támogatottsága és népszerűsége biztosítja, hogy az elkészült alkalmazások hosszú távon is fenntarthatók legyenek.

Fejlesztés során a natív Java nyelvet használtam az Android alkalmazásom működésének biztosítására. Ez magában foglalta a felhasználói interakciók kezelését, az adatbázis-műveletek megvalósítását és az alkalmazás logikájának kialakítását, miközben szorosan együttműködött az XML-ben definiált vizuális elemekkel. A Java nyújtotta rugalmasság és teljesítmény lehetővé tette, hogy az alkalmazás gyorsan és hatékonyan tudjon működni.

3.3. Firebase

A Firebase egy felhőalapú platform, amely a modern alkalmazásfejlesztéshez nyújt eszközöket és szolgáltatásokat. Fő célja, hogy megkönnyítse a fejlesztők számára az alkalmazások backend-funkcióinak megvalósítását, mint például az adatok tárolása, a felhasználók hitelesítése vagy értesítések küldése. Alapvető funkciói ingyenesek, azonban egy komplexebb alkalmazáshoz tudjuk bővíteni csomagunkat extra funkciók eléréséhez.

Fejlesztés során a Firebase szolgáltatásait használtam az adatok tárolására és szinkronizálására, valamint a felhasználók hitelesítésére. A Firebase rugalmassága és integrált megoldásai nagyban megkönnyítették az alkalmazásom fejlesztését és skálázhatóságának biztosítását.

3.3.1. Cloud Firestore

A Firebase egyik alapvető eleme a Firestore, egy NoSQL alapú adatbázis, amely valós idejű adatkezelést és szinkronizációt tesz lehetővé a felhasználói eszközök között. Támogatja az adatok összetett lekérdezését és a tranzakciókat, így könnyen kezelhetők bonyolult adatkapcsolatok is. Egyszerű szabályokkal lehet korlátozni az adatokhoz való hozzáférést, ezzel növelve a biztonságot különösebb erőfeszítés és időráfordítás nélkül.

3.3.2. Firebase Authentication

A Firebase Authentication a Firebase által nyújtott hitelesítési szolgáltatás, amely egyszerű és biztonságos megoldást kínál felhasználók azonosítására különböző platformokon. A célja, hogy minimalizálja a hitelesítéshez szükséges fejlesztési időt, miközben erős

biztonságot nyújt.

A szolgáltatás szorosan integrálható más Firebase megoldásokkal, például a Firestore-val, amely lehetővé teszi, hogy a hitelesített felhasználókhöz kapcsolódó adatok gyorsan és biztonságosan tárolhatók legyenek.

3.3.3. Cloud Storage

A Firebase Cloud Storage egy megbízható, skálázható felhőtárhely, amely fájlok, például képek vagy videók tárolására és elérésére szolgál. Egyszerűen integrálható alkalmazásokba. Én a képek tárolására használtam.

3.4. Room Database

A Room Database egy Android fejlesztésben használt könyvtár, amely leegyszerűsíti a helyi SQLite adatbázisok használatát. Elsődleges célja, hogy könnyen kezelhető, biztonságos és hatékony adatbázis-műveleteket biztosítson az alkalmazások számára, miközben automatikusan kezeli a komplex adatkapcsolatokat.

A Room három fő összetevőre épül: az entitások az adatbázis tábláit reprezentálják, a DAO-k (Data Access Objects) az adatbázis-műveleteket definiálják, a Database osztály pedig összefogja az adatbázis konfigurációját és hozzáférést biztosít. A Room támogatja az adatok típusbiztonságát, a reaktív programozást LiveData használatával, és automatikusan kezeli a migrációkat, ha az adatbázis struktúrája megváltozik.

A Room a háttér munkák kezeléséhez optimalizált, és lehetővé teszi az adatok gyors, hatékony és valós idejű lekérdezését, módosítását. Fejlesztés során a Room Database-t használtam az alkalmazásom helyi, offline adatkezelésének biztosítására, amely stabil és jól strukturált módon tárolta és kezelte az adatokat.

3.5. Egyebek

A diagramokat, mátrixokat a draw.io oldal segítségével készítettem. Itt több szoftverfejlesztéshez kapcsolódó diagram sablonja elérhető, valamint előre elkészített elemeket is biztosít.

A fejlesztést az Android Studio fejlesztői környezetben végeztem, ami az Android alkalmazások fejlesztésére szolgáló hivatalos integrált fejlesztőkörnyezet. A JetBrains IntelliJ IDEA platformjára épül. Intelligens kódszerkesztés, vizuális elrendezés szerkesztő, beépített emulátor, hibakereső és még sok más funkciója könnyíti a fejlesztést. Az Android Studio segítségével a fejlesztők gyorsan készíthetnek, tesztelhetnek és optimalizálhatnak alkalmazásokat különböző eszközökre és képernyőméretekre.

A GitHub-ot egyaránt használtam verziókövetésre és a feladatok kezelésére is. A különböző feladatrészekhez Issue-kat vettem fel. Amikor elkészültem egy feladattal, zároltam, így láttam milyen teendőim vannak még hátra és mikkel vagyok kész.

A Microsoft Excel egy népszerű táblázatkezelő szoftver, amit adatok rendszerezésére, elemzésére és vizualizálására használnak. Az alkalmazás elkészítése során táblázatok készítéséhez lett alkalmazva, például a piackutatás eredményének összegyűjtésére.

4. Architektúra

A Magyarországi Madarak mobilalkalmazás architektúrája két részből, a frontendből és a backendből áll. A frontendet XML segítségével valósítottam meg, míg a backendhez Java-t, Room Database-t és Firebase-t használtam.

4.1. Frontend

A frontend az a része az alkalmazásnak, amivel a felhasználó interakcióba lép. Ide tartoznak az olyan elemek, mint az alsó navigációs sáv, a madarak listázása, megfigyelések létrehozása, az autentikációs felület és még sok más is.

Az XML határozza meg a struktúrát és az oldal elemeit, elrendezésüket és stílusukat. Hasonló felépítésű és célú elemekhez sablonok készíthetők, így jelentősen csökkentve a kódDuplikációt, valamint egységes kinézetet ad az alkalmazásnak. A mögöttes logika nem része az XML-nek, így a gombok és beviteli mezők kezelése sem tartozik a feladatai közé.

4.2. Backend

A backend az a része az alkalmazásnak, ami a működést, a mögöttes logikát határozza meg. Ezen szerepkörbe tartozik az események kezelése, gombok és beviteli mezők működésének leírása, valamint az adatok kezelése és tárolása.

A fő része a natív Java, ami a jelentős részét kezeli az eseményeknek. Feldolgozza a felhasználói interakciókból keletkező adatokat, így a gombok nyomását, beviteli mezők tartalmának kezelését, és a megfelelő adatok küldését a frontendre.

A helyi adattároláshoz Room Database lett igénybe véve, ami nem tekinthető hagyományos backendnek. A könyvtár egyedi annotációkat tartalmaz, ami extra logikával egészíti ki az osztályokat és változókat, valamint a backend ezen részét nem kell implementálni, helyette a Room Database által biztosított szolgáltatások és metódusok alkalmazhatóak. A könyvtár egy biztonságos adatbázist is biztosít, így könnyítve a fejlesztők munkáját, növelve az alkalmazás értékét.

A Firebase sem tekinthető hagyományos backendnek, erősen hasonlít a Room Database-hez. Ez azt jelenti, hogy a hagyományos megoldással ellentétben itt sem szükséges külön backendet implementálni, helyette a Firebase által biztosított szolgáltatások és metódusok használhatók az autentikációk kezelésére, adatok tárolására és lekérésére. A Firebase is egy nagyon megbízható, biztonságos, könnyen kezelhető adatbázist nyújt.

5. Belső felépítés

5.1. Activity-k

Egy Android alkalmazás több activity-ből épül fel, amelyek az alkalmazás szorosan összetartozó részeit alkotják. Az activity-k egy adott képernyőhöz vagy funkcióhoz kapcsolódó feladatokat kezelnek, például egy bejelentkezési folyamatot vagy egy lista megjelenítését. Az egyes activity-k között navigációval biztosítható az alkalmazás logikai folyamata.

Az activity-k más komponensek (például fragmentek, service-ek) funkcióit is használhatják, és szorosan együttműködnek a vizuális elemekkel, amelyeket XML-ben definiálnak. Ez a felépítés lehetővé teszi az alkalmazás moduláris kialakítását és átláthatóbb fejlesztését. Minden activity osztályhoz tartozik egy megfelelő layout XML fájl is.

A Magyarországi Madarak mobilalkalmazásban az alábbi activity-k valósulnak meg:

– *KnowledgeBaseActivity*:

Ez a launcher activity az alkalmazásban, annyit jelent, hogy indításkor ez az oldal jelenik meg. Ez az úgynevezett főoldal, mely *Tudásbázis* néven szerepel a navigációban. Itt listázódnak a madarak és lehet keresni közöttük.

– *BirdPageActivity*:

Ez az oldal egy madárnak a részletező oldala. Itt kerül megjelenítésre egy faj minden jellemzője és a leírása. Megnyitásakor egy új activity-ként jelenik meg, tehát az activity, ahonnan megnyitottuk ezt az oldalt, még létezik, nem zárult be.

– *BirdIdentificationActivity*:

Ez a *Madárhatórozó* néven szereplő oldal. Itt három nagy szempont (tollazat színei, madár alakja, madár élőhelye) alapján kereshetünk madarakat. Fő célja, hogy egy látott madarat tudjunk könnyedén beazonosítani. Négy fragmentből épül fel, amik között gombok segítségével tudunk váltani. A fragmentek együtt osztozkodnak az activity életsiklusán, nagyobb lehetőséget adva a moduláris felépítésre. Négy fő fragmentből épül fel a *Madárhatórozó*:

- *BirdColorFragment*: A tollazat színeinek megadásához.
- *BirdShapeFragment*: A madár alakjának megadásához.
- *BirdHabitatFragment*: A madár élőhelyeinek megadásához.
- *BirdIdentificationResultsFragment*: Az eredmények megjelenítéséhez.

– *ObservationsActivity*:

Itt található a *Megfigyelések* oldal. Itt kerülnek listázásra a felhasználó megfigyelései, itt

lehet létrehozni, valamint innen érhetőek el a megfigyeléseket részletező oldal.

– *ObservationPageActivity*:

Ez egy megfigyelés részletező oldala, mely külön indított activity-ként jelenik meg. Ezt a sablont használja fel az alkalmazás megfigyelések készítésére, de már meglévő megfigyelések adatai is ide töltődnek be. Ezáltal itt lehet létrehozni, módosítani, törölni megfigyeléseket.

– *SettingsActivity*:

Ez az activity tartalmazza az alkalmazás beállításait, valamint a felhasználói beállításokat is, ha a felhasználó be van jelentkezve. Ha nincsen autentikált felhasználó, úgy innen érhető el a bejelentkezés és a regisztráció egy-egy gomb segítségével.

– *LoginActivity*:

A *Bejelentkezés* oldalon lehet bejelentkezni e-mail cím és jelszó megadásával vagy a Google bejelentkezés is választható. Innen érhető el a *Regisztráció* oldal is. Ez az oldal egy külön activity-ként indul el, amin a regisztráció oldalával osztozkodik.

– *RegisterActivity*:

A *Regisztráció* oldalon lehet regisztrálni e-mail cím és jelszó, valamint jelszó megerősítése beviteli mezők helyes kitöltésével. Regisztráláskor egy megerősítő e-mail kerül elküldésre. Ezen az activity-n elérhető még a Google bejelentkezés is, valamint a *Bejelentkezés* oldalra navigálás. Az az activity egy külön activity-ként jelenik meg, amin a *Bejelentkezés* oldalával osztozkodik.

5.2. Adapterek

Az adapterek az Android alkalmazásokban olyan osztályok, amelyek a különböző adatkészleteket és a felhasználói felület megjelenítéséért felelős elemeket kapcsolják össze. Az adapterek segítségével az adatok strukturáltan és dinamikusan jeleníthetők meg a képernyőn, miközben az egyes elemek megjelenését is testreszabhatjuk.

Az adapterek különféle forrásokból származó adatokat, például tömböket, listákat vagy adatbázisok lekérdezéseit, képesek kezelni, majd ezeket vizuálisan reprezentálják a megfelelő nézetekben, például RecyclerView-kban. Ez a felépítés lehetővé teszi a felhasználói felület és az adatok elkülönítését, így egyszerűbbé válik az alkalmazás fejlesztése és karbantartása.

A Magyarországi Madarak mobilalkalmazásban az alábbi adapterek szerepelnek:

– *ListBirdsAdapter*:

Az az adapter felelős azért, hogy lekérje a madarakat az adatbázisból, majd megjelenítse

azokat egy egyedi dizájnú listaszerkezetben. A lekérés és megjelenítés úgy van kialakítva, hogy ne akadályozza az UI szálát, így a felhasználó tudja használni az alkalmazást akkor is, ha még nincsen teljesen betöltve az összes elem. Az adatbázisváltozásokat is követi, így ha valami változik, akkor a megjelenítésben is változás történik. Ezt az adaptert a *Tudásbázis* és a *Madárhatározó* is használja.

– *BirdIdentificationAdapter*:

Ez az adapter egy egyedi kinézetű jelölőnégyzeteket felhasználva listáz adatokat. A *BirdIdentificationAdapter* a madár tollainak színeinek, madár alakjainak és madár élőhelyeinek listázásához lett felhasználva a *Madárhatározó* oldalon.

– *ObservationsAdapter*:

Ez az adapter kéri le az adatbázisból a felhasználó megfigyeléseit, majd megjeleníti azokat egy egyedi listanézetben. Ez az adapter a *Megfigyelések* oldalon került felhasználásra.

5.3. Adatok

A Magyarországi Madarak mobilalkalmazás követi a **követi az MVVM (Model-View-ViewModel)** architektúrát. Ez az androidfejlesztésben egy elterjedt mintázat, ami tiszta rétegződést biztosít, és különválasztja a különböző felelősségeket az alkalmazásban. Az adatok így Model, DAO, Repository, Database és ViewModel osztályokra és interface-kre lett felosztva. Ezek elsősorban a Room Database-hez lettek készítve és tervezve.

A Magyarországi Madarak mobilalkalmazásban az alábbi adatok készültek el:

– *Bird*:

Egy madár minden adata elérhető ebből az entitásból. Legtöbb adata kapcsolótáblák mentén tárolódnak, amit a Room Database automatikusan kezel. Emellett van egy *BirdEntity* típusú adattagja is, ami már ténylegesen is tartalmazza az olyan adatokat, amik nem referenciaként vannak megadva. A *BirdDAO* interface kezeli a keresztkapcsolattáblák mentén való összekötést, valamint a *Bird* és *BirdEntity* megfelelő létrehozását a lekérésekhez. A lekéréseket és szinkronizálást *BirdRepository* kezeli, ami a madarakhoz tartozó konstansok Firestore szinkronizálását is kezeli. A megjelenítésért a *BirdViewModel* felelős.

– *BirdEntity*:

Lényegében a Firestore-ban szereplő nem referenciaalapú adatokat tárolja, valamint a madár id-ját is, ami a Firestore-ban a dokumentum azonosítójaként van jelen.

– Konstansok:

Főként madarak adataihoz lettek létrehozva a redundancia elkerülése végett. Kapcsolótáblák segítségével kerülnek összeköttetésbe a madarakkal. Csupán egy id-ből, ami a Firestore-ban a dokumentum azonosítójaként szerepel, és egy értékből áll. Dao csupán a Firestore-ral való szinkronizáláshoz szükséges. A konstansok közé tartoznak:

- *ConservationValue*: A madarak természetvédelmi értékeinek meghatározásához. Egész számértéket tárol. Firestore szinkronizáláshoz *ConservationValueDAO* metódusai lettek használva.
- *Diet*: Madarak főbb táplálékait tárolja. Tartozik hozzá egy *DietCrossRef* is, mivel több-a-többhöz kapcsolat valósul meg közte és a madarak között. Firestore szinkronizáláshoz *DietDAO* metódusai lettek használva.
- *Color*: Egy madár tollainak színét tárolja. Tartozik hozzá egy *ColorCrossRef* is, mivel több-a-többhöz kapcsolat valósul meg közte és a madarak között. Firestore szinkronizáláshoz *ColorDAO* metódusai lettek használva.
- *Shape*: Egy madár lehetséges alakját tárolja. Tartozik hozzá egy *ShapeCrossRef* is, mivel több-a-többhöz kapcsolat valósul meg közte és a madarak között. Firestore szinkronizáláshoz *ShapeDAO* metódusai lettek használva.
- *Habitat*: Madarak főbb élőhelyét tárolja. Tartozik hozzá egy *HabitatCrossRef* is, mivel több-a-többhöz kapcsolat valósul meg közte és a madarak között. Firestore szinkronizáláshoz *HabitatDAO* metódusai lettek használva.

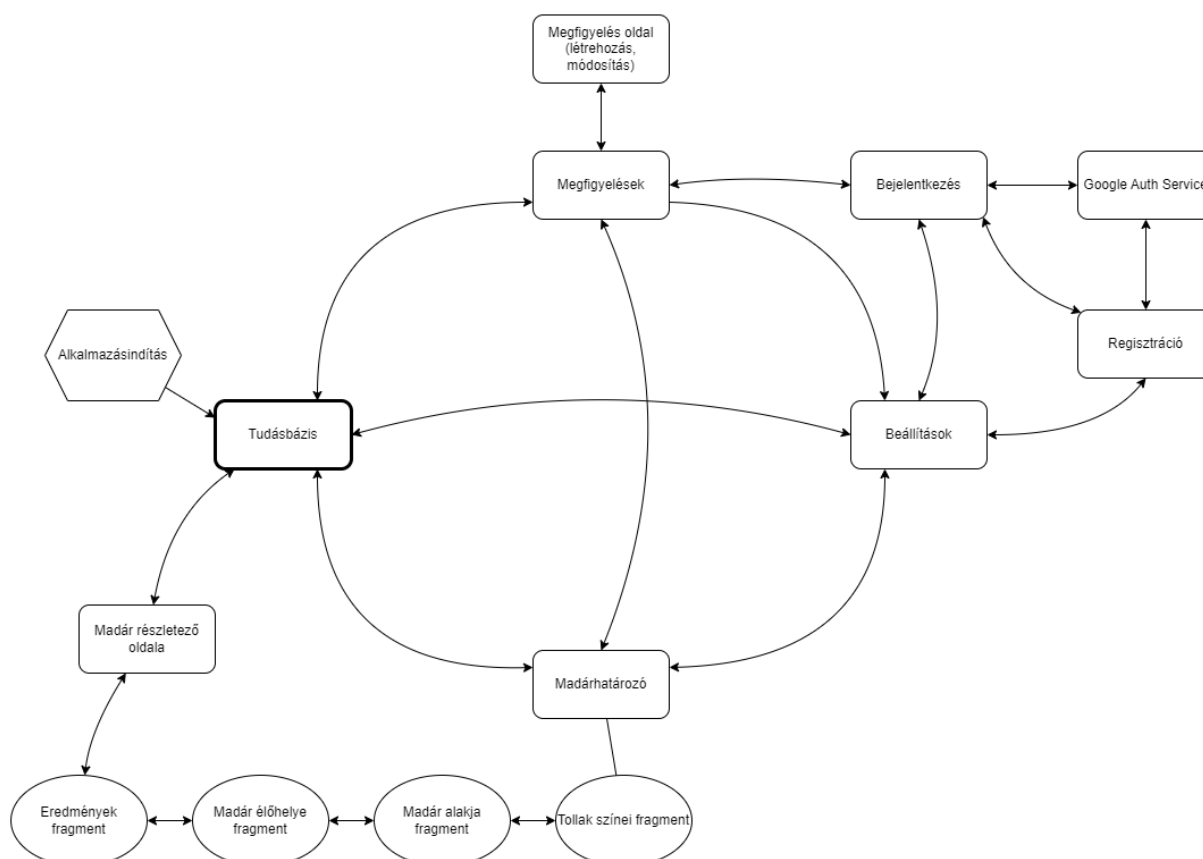
– *Observation*:

Egy megfigyeléshez tartozó összes adat tárolására szolgáló entitás. Firestore-ban tárolt adatoktól az *observationId*-ban tér el, ami a dokumentum azonosítójaként szerepel. A Room Database az *ObservationDAO* interface segítségével hajtja végre az adatbázisműveleteket. A felhővel való szinkronizálásért és az adatbázisműveletek megfelelő használatáért az *ObservationRepository* felelős. A megjelenítés és az UI-val való kommunikáció az *ObservationViewModel* feladata.

– *User*:

Mivel végül nem kerül tárolásra külön felhasználói adat, így végül nem lett használva.

5.4. Navigáció



5.4.1. ábra – Navigációs gráf

A Magyarországi Madarak mobilalkalmazásban a navigációt egy alsó navigációs sáv kezeli a főbb modulok eléréséhez. A főbb modulok közötti váltás során az activity megszűnik, majd elindul az az activity, ahova navigáltunk.

Az 5.4.1.-es ábrán látható, hogy az alkalmazás induló activity-je a *Tudásbázis*. A madarak részletes leíró oldalát megnyitva új activity indul a korábbi bezárása nélkül. Itt már nem érhető el a navigációs menü.

Érdekesség lehet, hogy a *Madárhatározó* oldal fragmenteket használ a tartalom megjelenítéséhez. Itt az alsó navigációs menü látható minden esetben. A fragmentek között előre és hátra gombok segítségével lehet váltani szekvenciálisan haladva. A legvégső eredmények fragmentben megjelenített madár találatokra kattintva megtekinthető a leíró oldaluk, ami egy új ablakban jelenik meg.

Végül kiemelném az autentikációt. Ha ki vagyunk jelentkezve, úgy a *Megfigyelések* oldal felajánlja a bejelentkezés lehetőségét, így elérhető lesz onnan is. Alapértelmezetten viszont a *Beállítások* oldalról érhető el a *Bejelentkezés* és *Regisztráció*. Mindkettő lehetőséget biztosít Google-os bejelentkezésre, ahol a Google külső bejelentkeztető oldala jelenik meg.

6. Biztonság

A Firestore lehetőséget biztosít biztonsági szabályok írására, ami korlátozhatja az adatbázisban tárolt adatokhoz való hozzáférést. A Security Rules szolgáltatás használatával megadtam, hogy egyes kollekciókban milyen szabályok legyenek érvényesek az adatok kezelése során külön olvasás, létrehozás, módosítás és törlés szinten.

C = létrehozás R = olvasás U = módosítás D = törlés	Felhasználó (saját)	Felhasználó (többi)	Madarak	Természetvédelmi értékek	Táplálékok	Színek	Alakok	Élőhelyek	Megfigyelések
Látogató	C		R	R	R	R	R	R	
Felhasználó	RUD		R	R	R	R	R	R	CRUD

6.1. ábra – Hozzáférési mátrix

A 6.1. ábrán láthatóak a látogatók és a bejelentkezett felhasználók jogai az egyes objektumokhoz vagy erőforrásokhoz való hozzáférés során. Ez a mátrix a Firestore adatbázisra vonatkozik.

Felhasználói fiók létrehozására csak látogatóknak van lehetőségük, valamint csak ők tudnak bejelentkezni.

Bejelentkezés után lehetőségünk van a fiókunkhoz tartozó adatok megtekintésére. Csak saját fiókot tudunk törölni és módosítani. Fontos kiemelni, hogy az alkalmazásban jelenleg nincs lehetőség kapcsolatba lépni, interaktálni más felhasználókkal, ezért más felhasználói adatokhoz nem tudunk hozzáférni semmilyen formában.

A természetvédelmi értékek, táplálékok, színek, alakok és élőhelyek amolyan konstans szereppel rendelkeznek, redundancia elkerülése végett lettek kiszervezve. Ezek olyan adatok, melyek nem változnak, és nem is bővülnek, így csak lekérni lehet. Ami érdekes lehet még az az, hogy a madarak is konstans jelleggel léteznek. Róluk is csak tájékozódni lehet, módosításra nincs lehetőség.

A megfigyelések jelölése is érdekes lehet. Ugyan a látogatók tudnak létrehozni, majd teljeskörűen rendelkezni felettük, de ez csak a helyi tárolásra vonatkozik. Csak a bejelentkezett felhasználók megfigyelései kerülnek be a felhőbe is, amolyan biztonsági mentésképpen. És nekik van lehetőségük így átvinni másik eszközre is megfigyeléseiket.

```

1 rules_version = '2';
2
3 service cloud.firestore {
4   match /databases/{database}/documents {
5     match /birds/{document} {
6       allow read: if true;
7     }
8
9     match /conservationValues/{document} {
10      allow read: if true;
11    }
12
13    match /diets/{document} {
14      allow read: if true;
15    }
16
17    match /colors/{document} {
18      allow read: if true;
19    }
20
21    match /shapes/{document} {
22      allow read: if true;
23    }
24
25    match /habitats/{document} {
26      allow read: if true;
27    }
28
29
30    match /observations/{observationId} {
31      allow read: if request.auth != null && resource.data.userId == request.auth.uid;
32      allow write: if request.auth != null && request.resource.data.userId == request.auth.uid;
33      allow delete: if request.auth != null && resource.data.userId == request.auth.uid;
34    }
35  }
36 }

```

6.2. ábra – Hozzáférési szabályok

A 6.2. ábra megmutatja a teljes szabálykészletet, ami a Firebase-ra vonatkozik. Az *allow read: if true;* sorok engedélyezik a lekéréseket, olvasásra lekérhetőek ezek az elemek. Írásra azonban nem soroltam fel a szabályokat (*allow read: if false;*), mivel ez az alapértelmezett beállítás, ha valamire nem vonatkozik megszorítás. Ezek a szabályok biztosítják, hogy a madarakat, természetvédelmi értékeket, étrendeket, színeket, alakokat és élőhelyeket csakis kizárólag megtekinteni lehet, módosítani, létrehozni és törölni nem lehet.

Az *if request.auth != null* részek felelnek azért, hogy a megfigyelésekhez csak akkor kérhessük le, hozzassunk létre vagy módosíthassuk, valamint törölhessük, ha bejelentkezett, autentikált felhasználók vagyunk. A második fele a lekérdezésnek, tehát a *resource.data.userId == request.auth.uid*, biztosítja azt, hogy csak akkor férünk hozzá a műveletekhez, ha a megfigyelés ténylegesen hozzánk, az autentikációban szereplő felhasználóhoz tartozik. Ez biztosítja a felhőadatokhoz való hozzáférések korlátozását a hozzáférési mátrix (6.1. ábra) alapján.

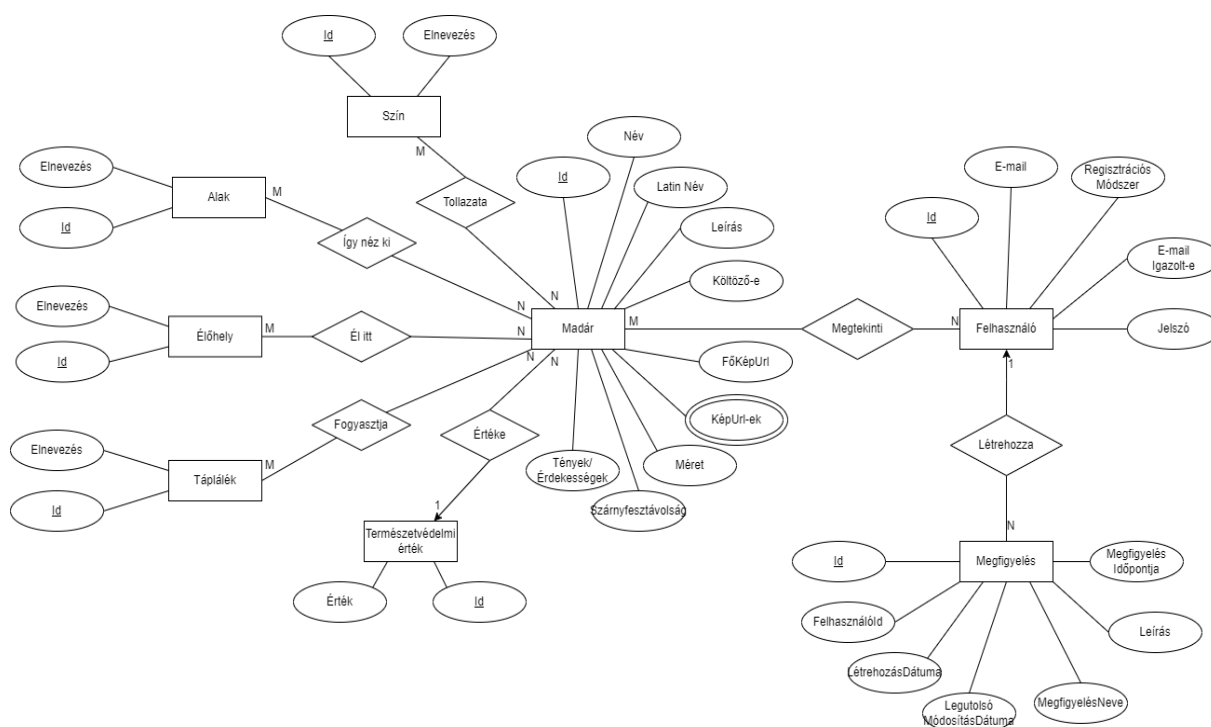
A helyi adattároláshoz Room Database-t használtam. Az Android alapértelmezett fájl-hozzáférési modellje, miszerint az adatokhoz alapvetően csak őket létrehozó alkalmazások férhetnek hozzá közvetlenül, elegendő biztonságot nyújthat.

A típusok ellenőrzését és konverziót az alkalmazás ellenőrzi.

7. Adatmodell

A Firestore szolgáltatását és a Room Database-t vettem igénybe az adatok tárolásához. A Firestore egy NoSQL adatbázis, ahol az objektumokat dokumentumok képviselik, amik kollekciókban tárolódnak. A Room Database pedig egy Androidon használt ORM (Object-Relational Mapping), amely az SQLite adatbázist egyszerűsíti, relációs struktúrát és típusbiztonságot biztosítva.

A két adatbázis hasonló felépítésű, de vannak apróbb eltérések. A Firestore lehetőséget nyújt referencia típusú adattagok tárolására. Olyan is előfordulhat, hogy egy listában vannak ezek a referenciák eltárolva, így ezekhez nem kell külön kapcsolótáblát létrehozni. A Room Database viszont csak egyszerű típusokat tárol, így kellett hozzá létrehozni kapcsolótáblákat is. További eltérés, hogy a Firestore képes dátumot is tárolni, amire a Room Database szintén nem képes. Ezekhez külön TypeConverter-eket kellett létrehozni, ami egyszerű szöveggé alakítja megadott szabály szerint a komplexebb típusokat. Eltérés van a felhasználói adatok tárolása terén is. Például helyileg nem tárolunk felhasználóról információkat, az kizárólag a felhőben tárolódnak a Firestore és Firebase Authentication jóvoltából.



7.1. ábra – Egyed-kapcsolat diagram

A 7.1. ábrán látható a rendszer egyed-kapcsolat diagramja. Kiemelném, hogy az ábrán feltüntetett kép url-ekhez tartozó képek a felhőben a Firebase Storage-ban lettek eltárolva. Offline is elérhetőek, mivel néhány képet az eszközön is eltárol.

7.1. Felhasználó

A felhasználók regisztrálást követően jönnek létre. A hozzájuk tartozó adatok többségét csupán a Firebase Authentication tárolja, biztonsági okokból nem kerülnek külön tárolásra az eszközön. A diagramra (7.1. ábra) a számunkra fontosabb adatok kerültek fel.

Érdekesebb adatok között szerepelhet a jelszó, amit alapvetően titkosít a Firebase saját magának.

A regisztrációs módszer alapján lehet beazonosítani, hogy valaki e-mail és jelszó segítségével regisztrált-e vagy Google-val. Ez alapján jelenít meg bizonyos mezőket. Például a beállításokban jelszó módosítására csak akkor van lehetőség, ha azzal regisztráltál. Valamint a fiók törléséhez is csak akkor kell megadni a régi jelszót.

Szerepel a diagramon az *e-mail igazolt-e*. Ez azt a célt szolgálja, hogy ha e-mail és jelszóval regisztrálsz, csak akkor vált igazra, ha a kiküldött megerősítő e-mailben lévő linket meglátogatja a felhasználó. Amíg nincs megerősítve az e-mail, nem enged bejelentkezni sem.

7.2. Megfigyelés

A felhasználók tudnak megfigyeléseket létrehozni. Ilyenkor a megadott adatok, mint a megfigyelés neve, időpontja és leírása kerülnek bírásra, a többi adatot automatikusan állítja be az alkalmazás.

A felhasználóId mező ellenőrzése biztosítja, hogy csak a saját megfigyelésünkhöz tudjunk hozzáférni. Ezért is szerepel 1:N kapcsolat a felhasználó és megfigyelés között, mivel egy felhasználó bármennyi megfigyelést létrehozhat, de csak egy felhasználóhoz tartozhatnak.

A legutolsó módosítás dátuma alapján kerül rendezésre a megfigyelések listázása. A létrehozás dátuma alapján pedig be tudjuk azonosítani, hogy mikor készült a megfigyelésünk.

7.3. Madár

Madarakat is egyszerűen id alapján azonosíthatunk be, ami a Firestore-ban a dokumentum azonosítója. A különféle fajokról rengeteg információt eltárolunk.

A természetvédelmi érték egy külső kulcsként szerepel a madárban és a kapcsolat mentén azonosítható be.

A táplálékok, színek, alakok, élőhelyek N:M kapcsolata miatt eltér a Firebase és Room Database tárolás szempontjából. A Firebase enged tömbben, listában tárolni hivatkozásokat, így ott egyszerű ezt kivitelezni. A Room Database-ban nincs ilyesmire lehetőség. A több-á-többhöz kapcsolatot kapcsolótáblával kell feloldani, ahova bekerül a madár azonosítója, valamint a listában szereplő értékek egyenként, és így alkotnak egy rekordot.

7.4. Természetvédelmi érték

A madarak természetvédelmi értéke sokszor megegyezik, így a redundancia elkerülése végett ki lettek szervezve külön táblába. Egy rekord vagy dokumentum egy azonosítót tartalmaz (dokumentum esetében ez a dokumentum azonosító), valamint egy egész értéket, ami az értékét jelöli. A nulla számérték azt jelöli, hogy a madár egyáltalán nem védett.

7.5. Táplálék

A madarak táplálékai redundancia elkerülése végett ki lettek szervezve külön táblába. Egy rekord vagy dokumentum egy azonosítót tartalmaz, valamint egy szöveget, ami a konkrétan megnevezett táplálék.

7.5. Szín

A madarak tollainak színei redundancia elkerülése végett ki lettek szervezve külön táblába. Egy rekord vagy dokumentum egy azonosítót tartalmaz, valamint egy szöveget, ami a konkrétan megnevezett szín.

7.5. Alak

A madarak lehetséges alakjai redundancia elkerülése végett ki lettek szervezve külön táblába. Egy rekord vagy dokumentum egy azonosítót tartalmaz, valamint egy szöveget, ami a konkrétan megnevezett alak.

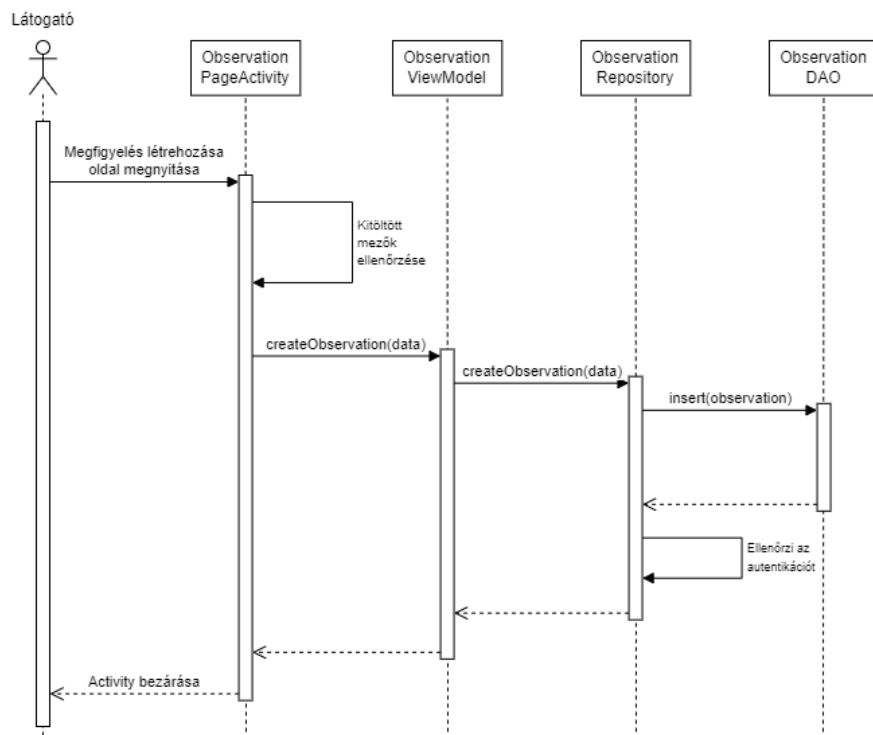
7.5. Élőhely

A madarak élőhelyei redundancia elkerülése végett ki lettek szervezve külön táblába. Egy rekord vagy dokumentum egy azonosítót tartalmaz, valamint egy szöveget, ami a konkrétan megnevezett élőhely.

8. A rendszer magasszintű folyamatai, működése

A Magyarországi Madarak mobilalkalmazás sok apró folyamatra épül. Mivel egy **MVVM (Model-View-ViewModel)** architektúrát követ, ami tiszta rétegződést biztosít, és különválasztja a különböző felelőségeket az alkalmazásban, a folyamatok tiszták, könnyen érthetőek és követhetőek.

Az alkalmazás egyik magasszintű folyamata a megfigyelés létrehozása, ahogy ez az egyik fő funkciója is. Ebben a fejezetben ezt a folyamatot fogom részletezni és ezen keresztül fog látszódni az alkalmazás rétegződése is.



8.1. ábra – Szekvencia diagram megfigyelés létrehozásáról

A 8.1. ábra mutatja be egy megfigyelés létrehozásának szekvencia diagramját. Ez a megfigyelések oldal jobb alsó sarkában található plusz ikonra kattintva nyitható meg.

Miután az alkalmazás betöltötte az oldalt, el lehet kezdeni az oldal kitöltését. Az első beviteli mező a megfigyelés neve, második helyen a megfigyelés időpontja található és legvégül a megfigyelés leírása. Az időpont alapvetően a megnyitáskori időpillanatra van beállítva, és a leírás csupán opcionális.

Alul a mentés gombra kattintva a rendszer ellenőrzi a megadott adatokat és jelez, ha a felhasználónak, ha valami probléma merült fel a mezőkben megadott adatokat illetően.

Ha a rendszer megfelelőnek találta a bevitt adatokat, úgy elkezdődik a létrehozásának folyamata. A rendszer meghívja az ObservationViewModel-nek a *createObservation()* metódusát, paraméterben átadva a felhasználó által megadott adatokkal.

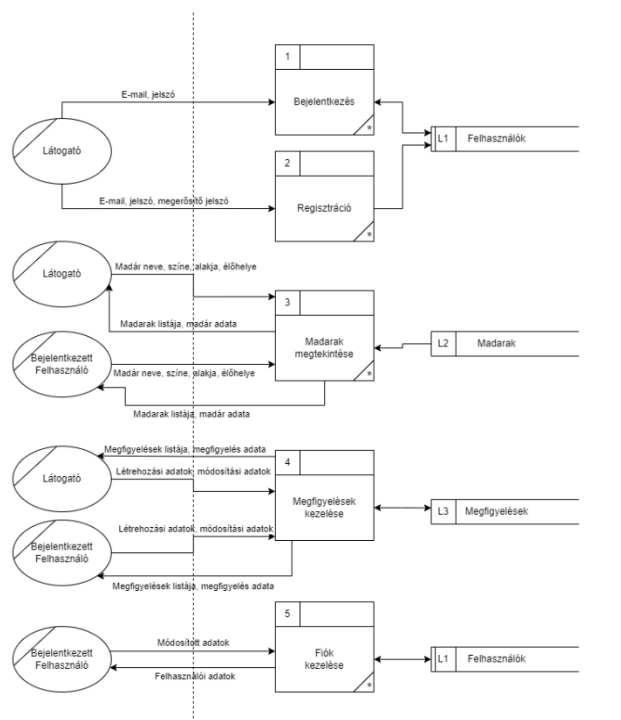
Az ObservationViewModel köti össze az UI-t és a Repository-t. A meghívott metódusban csupán annyi történik, hogy meghívja az ObservationRepository-nak a *createObservation()* metódusát, aminek átadja paraméterben a kapott adatokat, valamint autentikációtól függően a felhasználó azonosítóját, vagy egy "local" kulcsszót, ami jelzi, hogy ez csak helyileg létezik.

Az ObservationRepository-nak van hozzáférése az observationDAO-hoz. Itt a művelethez indít az alkalmazás egy külön szálát, hogy ne foglalja le az alkalmazás fő szálát. Ezt követően létrehoz egy megfigyelés példányt. A létrehozás időpontjának és a legutolsó módosítás dátumának is beállítja a jelenlegi időpillanatot. A paraméterben kapott adatokat átadja az új objektumnak, valamint hozzárendel egy egyedi azonosítót is. Ezt követően meghívja az observationDAO *insert()* metódusát, átadva a létrehozott metódust paraméterben.

A DAO végrehajtja a műveletet, beszúrva a megfigyelést a helyi adatbázisba.

A Repository ezt követően leellenőrzi, hogy autentikált felhasználók vagyunk-e, hogy beszúrja-e a felhőbe is, de ez egy összetettebb folyamat része.

Végül véget ér a Repository metódusa, majd a ViewModel is és visszaérünk az ObservationPageActivity-re. Itt meghívódik a *finish()* metódus és véget ér az activity. Visszatérünk az ObservationActivity-re.

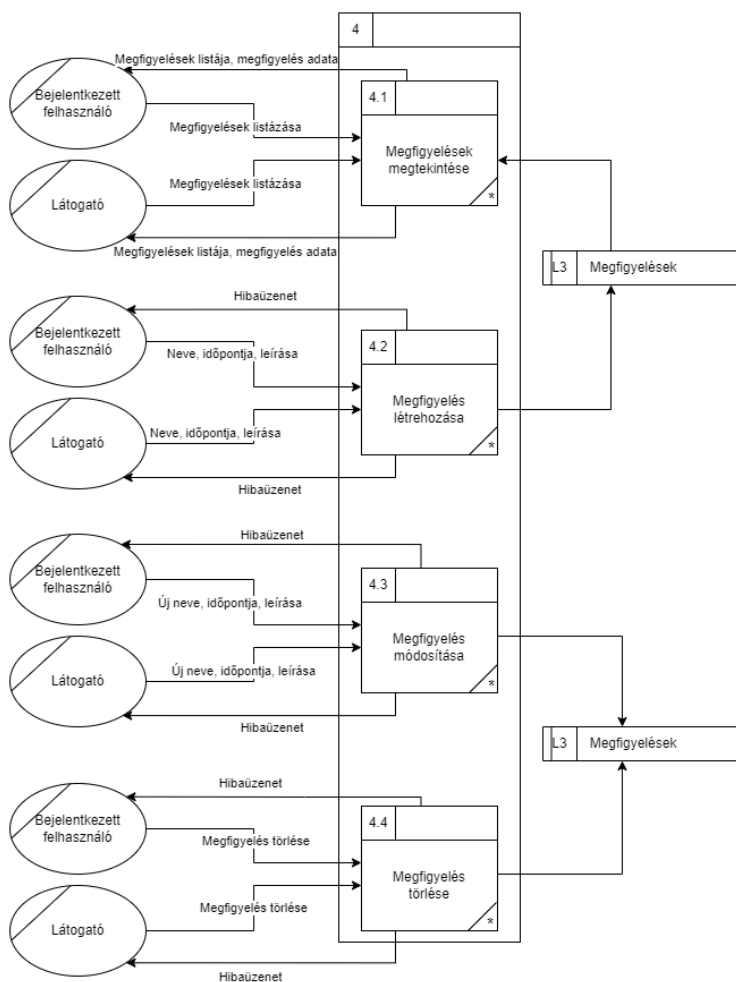


8.2. ábra – Logikai adatfolyam diagram, első réteg

A 8.2. ábrán látható egy logikai adatfolyam diagram első rétege, ami ezúttal a teljes rendszert írja le. Az átláthatóság miatt az egyes elemeket többször is felvettük. Az ábrán látható, hogy a folyamatok többsége egyszerű, a legösszetettebb közülük a *Megfigyelések kezelése*.

A diagram fontos tényezője, hogy látható, a folyamatok mely adattárakkal kommunikálnak és milyen módon. Egyes helyeken csak olvassuk azokat, van olyan is, ahol csak írni kell az adattárakba.

Egy megfigyelés kezelése összetett folyamat, ezért részfolyamokra bontható. Erről egy második szintű adatfolyam diagramot is készítettem, ami megtekinthető a 8.3. ábrán. Az összetett folyamat közé tartozik a megfigyelések megtekintése, megfigyelés létrehozása, módosítása és törlése. Ezekre mindegyik felhasználó képes, azonban itt is ki kell emelni, hogy a látogatók megfigyelései nem szinkronizálódnak a felhőbe.



8.3. ábra – Logikai adatfolyam diagram, második réteg

9. Fontosabb kódrészletek ismertetése

9.1. Konstansok szinkronizálása

Az egyik érdekes kódrészlet ahhoz tartozik, hogy hogyan szinkronizálódnak a konstans adatok a Firestore és Room Database között. Az összes konstans madarat kiegészítő érték a 9.1.1. ábrán bemutatott *syncFirestoreColors* függvény mintájára működik, így ennek a megtekintésével egy nagyobb kódrészletet érhetünk el.

```
private static void syncFirestoreColors() { 1 usage
    FirebaseFirestore firestore = FirebaseFirestore.getInstance();

    firestore.collection( collectionPath: "colors") CollectionReference
        .get() Task<QuerySnapshot>
        .addOnSuccessListener(queryDocumentSnapshots -> {
            for (DocumentSnapshot document: queryDocumentSnapshots) {
                Color color = new Color(document.getId(), Objects.requireNonNull(document.getString( field: "colorName")));
                Executors.newSingleThreadExecutor().execute(() -> {
                    instance.colorDAO().insert(color);
                    Log.d( tag: "DATA", msg: "--Firestore color got: " + color.getColorName() + ".--");
                });
            }
        }).addOnFailureListener(e -> {
            Log.e( tag: "DATA", msg: "--Failed to load colors from FireBase.--", e);
        });
}
```

9.1.1. ábra – syncFirestoreColors függvény

A szinkronizáló függvény meghívásakor elkérünk egy Firestore példányt. Ezt követően megadjuk, hogy melyik kollekciónak szeretnénk adatokat lekérni, jelen esetben ugye a „colors” kollekciónak. Miután sikerült lekérni az elemeket a kollekciónak, bejárjuk egyesével egy ciklus segítségével a dokumentumokat. Konvertáljuk őket a megfelelő objektummá, itt a példában ez Color objektumot jelent.

Ezt követően az Executors segítségével indítunk egy új szálát, hogy ne a fő UI szálát blokkoljuk. Eerre azért van szükség, mivel ugyan a Firestore lekérdezés aszinkron művelet, de mivel egy külső feladatra mutatunk feldolgozás közben, így alapvetően blokkolná az UI szálát. A külön szálunkban beszúrjuk a lekért rekordunkat a Room Database-ba a ColorDAO *insert* műveletével.

9.2. Referenciák lekérése

A következő kódrészletet azért emelném ki, mivel alapvetően nem gondolnánk mennyi extra lépést igényel a tömbökben eltárolt referencia típusú adatok lekérése és kezelése. Úgy gondolom ez a rövid, elegáns kódrészlet kiválóan ellátja ezt a feladatot. ábrán is látható.

```

private static List<String> getFirestoreReferences(List<DocumentReference> references) { 4 usages
    if (references != null) {
        return references.stream() Stream<DocumentReference>
            .filter(Objects::nonNull)
            .map(reference -> reference.getPath().substring( beginIndex: reference.getPath().lastIndexOf( ch: '/' ) + 1))
            .collect(Collectors.toList());
    } else {
        return null;
    }
}

```

9.2.1. ábra – getAllForOneUser függvény csoportok lekérésére

Először is ellenőrizzük, hogy tényleg kaptunk-e értékeket. Ha nem kaptunk, akkor nem tudunk értelmes adatot visszaadni.

Ezt követően a kapott listánkat streammé, egyfajta csővezetékké alakítjuk, ami egy hatékony módja az adathalmazok feldolgozásának.

A *filter* segítségével kiszűrjük a null értékeket a listánkból. A null értékekkel nem tudunk mit kezdeni.

Ezt követően a *map* segítségével feldolgozunk minden elemet. Itt minden elem már DocumentReference típusúként szerepel, így tudjuk venni az elérési útvonalukat a *getPath* meghívásával. Ennek vesszük az ebből az útvonalból lekért utolsó „/” után lévő szakaszt, mivel ez lesz a dokumentumunk id-ja egyszerű szöveggént.

Legvégül összegyűjtjük ezeket az elemeket és visszaadjuk a dokumentum azonosítókat, tehát kapunk egy egyszerű szövegeket tartalmazó listát. A visszaadott értékek segítségével már egyszerűen le tudjuk kérni az adatokat a referenciák mentén a Firestore-ból. Ehhez csak végig kell járni a listát például egy ciklus segítségével, és lekérni Firestore-ból a dokumentumokat *firestore.collection(collectionName).document(id).get()* formátumban.

9.3. Típuskonverterek

Room Database használatakor beleütköztem abba a problémába, hogy csupán egyszerű típusokat képes tárolni. Szerencsére biztosít könnyen használható eszközt ennek kiküszöbölésére, az úgy nevezett típuskonvertereket. Ezek a ConverterUtils-ban találhatóak.

```

@TypeConverter 4 usages
public static String fromListToString(List<String> list) {
    return new Gson().toJson(list);
}

@TypeConverter 8 usages
public static List<String> toListFromString(String s) {
    return new Gson().fromJson(s, new TypeToken<List<String>>() {}.getType());
}

```

9.3.1. ábra – Típuskonverterek

A „*@TypeConverter*” annotációt a Room Database szolgáltatja. Az ilyen jelölésű statikus metódusokat át lehet adni az adatbázisnak, ami minden esetben, amikor valamilyen problémába ütközik, amikor megpróbál adatot beszűrni, típuskonverterekhez nyúl.

A *fromListToString* metódus paraméterben egy szövegekből álló listát vár és visszaad egyetlen szöveget, amit a *toJson* segítségével tehetünk meg.

A *toListFromString* metódus egy szöveget vár és ebből hoz létre egy szövegeket tároló listát *fromJson* segítségével. Ennek a metódusnak meg kell adnunk a szöveget, valamint azt, hogy milyen típust keresünk a szövegben. Ha megtalálja a „*List<String>*”-et, képes lesz visszaalakítani a szöveget listává. Végül visszaadja ezt az eredményt.

Elsőre bonyolultnak tűnik ugyan, de ha belegondolunk elég egyszerű, hasznos kis metódus hozhatóak létre gyorsan.

9.4. Szűrt eredmények frissítése

Ez a folyamat a *BirdIdentificationResultsFragment*-en belül történik. Ezt a metódust azért emelném ki, mert komoly fejtörőt okozott a működőképes verziójának elkészítése. A kód felépítését tekintve több helyen ismétlődik, így csak egy részét mutatnám be.

```
private void updateFilteredResults(List<Bird> allBirds) { 4 usages
    List<Bird> filteredBirds = new ArrayList<>(allBirds);

    List<Color> currentColors = getCurrentColors();
    if (!currentColors.isEmpty()) {
        filteredBirds = filteredBirds.stream()
            .filter(bird -> new HashSet<>(bird.getColors()).containsAll(currentColors))
            .collect(Collectors.toList());
    }

    List<Shape> currentShapes = getCurrentShapes();
    if (!currentShapes.isEmpty()) {
        filteredBirds = filteredBirds.stream()
            .filter(bird -> new HashSet<>(bird.getShapes()).containsAll(currentShapes))
            .collect(Collectors.toList());
    }

    List<Habitat> currentHabitats = getCurrentHabitats();
    if (!currentHabitats.isEmpty()) {
        filteredBirds = filteredBirds.stream()
            .filter(bird -> new HashSet<>(bird.getHabitats()).containsAll(currentHabitats))
            .collect(Collectors.toList());
    }

    birdIdentificationResults.setValue(filteredBirds);
}
```

9.4.1. ábra – *updateFilteredResults* metódus

A metódus célja az, hogy a Madárhatározóban kiválasztott jellemzők szerint kigyűjtse a madarakat, majd beállítsa azokat. Nézzük meg a színeknél mi történik. Először is paraméterben megkapjuk az összes lehetséges madarat, amiből szűrhetünk. Ezt követően érünk el a színek szűréséhez. Itt lekérjük egy listába a kiválasztott színeket. Ha nincs

kiválasztott szín, akkor az összes madarat megtarthatjuk szűrés nélkül.

Ha van kiválasztott színünk, akkor indítunk egy streamet, egy csővezetékét. Itt kihasználjuk, hogy különböző elemeink vannak, és átalakítjuk a színeket tartalmazó listánkat egy halmazzá, amelyik adatszerkezetnek van beépített függvénye a teljes tartalmazásra („containsAll”). Tehát itt minden olyan madarat választunk ki, amelyik tartalmazza legalább az összes olyan színt, ami a listában szerepel. Ezeken felül is tartalmazhatnak még színeket. A kiválasztott elemeket végül összegyűjtjük egy listába.

Ezen a szűrt listán végrehajtjuk majd az alakszűrést is és az előhelyszűrést is, és beállítjuk a kész, szűrt listát az eredményeknek.

9.5. Elérhető-e az Internet

A következő kódrészlet azt mutatja be, hogyan lehet egyszerűen ellenőrizni, hogy csatlakozva vagyunk-e az Internethez.

```
public static boolean isInternetAvailable(Context context) { 2 usages
    ConnectivityManager connectivityManager =
        (ConnectivityManager) context.getSystemService(Context.CONNECTIVITY_SERVICE);
    NetworkInfo activeNetwork = connectivityManager.getActiveNetworkInfo();
    if (activeNetwork != null) {
        Log.i(LOG_TAG, msg: "--Internet available.--");
        return true;
    }
    Log.w(LOG_TAG, msg: "--Internet unavailable.--");
    return false;
}
```

9.5.1. ábra – isInternetAvailable metódus

Ebben a metódusban először is elkérünk egy ConnectivityManager-t. A Managertől lekérjük a hálózati státuszt. Ha ez *null* érték, akkor nem érhető el az Internet és *false*-t visszaadhatunk.

Ha viszont nem *null* az értékünk, úgy csatlakozva vagyunk az Internethez és *true*-t adunk vissza.

Megjegyezném, hogy volt egy olyan verzió is, ahol azt is ellenőriztük, hogy Wi-fi-re vagyunk-e csatlakozva, hogy ne fogyasszuk az értékes mobilnetet. Ez végül el lett vetve, mert az alkalmazást sokszor olyan helyen is használnánk, ahol csak mobilinternet elérhető.

9.6. Autentikált-e a felhasználó

Sokszor van arra szükségünk, hogy leellenőrizzük, hogy hozzá férhet-e bizonyos funkciókhoz a felhasználó. Erre a célra lett kialakítva a 9.6.1. ábrán látható metódus. Ám nem elég csak azt figyelni be vagyunk-e jelentkezve. A metódust az AuthUtils tartalmazza.

```

public static boolean isUserAuthenticated() { 18 usages
    FirebaseUser currentUser = FirebaseAuth.getInstance().getCurrentUser();

    if (currentUser != null) {
        currentUser.reload().addOnCompleteListener(task -> {
            if (task.isSuccessful()) {
                if (currentUser.isEmailVerified()) {
                    Log.i(LOG_TAG, msg: "--User authenticated and email verified.--");
                } else {
                    Log.w(LOG_TAG, msg: "--User email still not verified.--");
                }
            } else {
                Log.e(LOG_TAG, msg: "--Failed to reload user data.--");
            }
        });
        return currentUser.isEmailVerified();
    }

    Log.w(LOG_TAG, msg: "--No authenticated user detected.--");
    return false;
}

```

9.6.1. ábra – isUserAuthenticated metódus

Legelőször megnézzük van-e bejelentkezett felhasználó. Amennyiben nincs, úgy nem is lehet autentikált a felhasználó.

Ezt követően újrahitelesítjük a felhasználót a Firebase-ban. Ehhez nem kell a felhasználó, automatikusan megy végbe. Egyébként erre azért van szükség, mert lehet, hogy a felhasználói fiókunk még nem hiteles, amikor lekérjük, és menet közben megváltozik. Mivel már le van kérve egy változóba ekkorra, így nem is változik a Firebase-ban található adatokkal együtt.

Miután sikeresen újratöltöttük a felhasználót, úgy megnézzük, hogy az e-mail címe hitelesítve van-e. E-mail és jelszó regisztrációs módszerrel kiküldésre kerül egy megerősítő e-mail üzenet is. Az üzenetben lévő linkre kattintva hitelesíthetjük az e-mail címünket. Tehát amikor még nincsen megerősítve, úgy a fiókunk előnyeit még nem élvezhetjük.

Érdekelhet minket, hogy mi történik azokkal a felhasználókkal, akik nem e-mail és jelszó segítségével regisztráltak, hanem Google által. Elég egyszerű a válasz: náluk alapvetően meg van erősítve az e-mail címük és ezt a tényt a Google biztosítja nekünk. Így a Firebase automatikusan hitelesítettnek veszi az ilyen felhasználókat.

Összegzésül akkor adunk vissza igazat arra, hogy autentikált-e a felhasználó, ha van egyáltalán felhasználónk, és ha az újratöltést követően meg van erősítve az e-mail címe.

10. Tapasztalatok, továbbfejlesztési lehetőségek

10.1. Alkalmazás megtervezése

A Magyarországi Madarak mobilalkalmazás megtervezésére nem volt túl sok időm. Eleinte többségében papírra vettem le gondolataimat és ötleteimet, amiből akadt bőven. Ez sokak számára kontraproduktívnak tűnhet, de olyankor is tudtam vele foglalkozni, amikor nem volt lehetőségem elektronikus eszközök használatára, vagy csak korlátozottan voltak elérhetőek.

Tervezés során sikerült olyan részletességgel kidolgoznom a különböző részeket, hogy amikor nekikezdtem a kódolásnak, már ott lebegett a szemem előtt a kész modul, és ezt próbáltam meg minél pontosabban megvalósítani, ami extra motiválóerővel is bírt. Ez segített abban, hogy szüntelenül tudjak foglalkozni a témával.

Meglátásom szerint viszont kellett volna több diagramot készítenem az elején, hogy a hibákat elkerülhessem. Sok idő ment el feleslegesen például azzal, hogy a Room Database adatbázis végső alakját elkészítsem.

10.2. Ütemterv, feladatok

A fejlesztés legelején nem volt különösebb ütemtervem. Lényegében csak azt mondtam magamnak, hogy minden nap foglalkozok a szakdolgozat elkészítésével. Ha éppen van más tantárgy, amivel foglalkozni kell, akkor kevesebbet, ha nincs ilyen, akkor több energiát teszek bele. És ehhez sikerült is tartanom magam, ami azért volt meglepő, mert végig motivált tudtam maradni a témával kapcsolatban a rengeteg szünet nélküli foglalkozás ellenére.

A projekt kezdetén próbáltam GitHub issue-kat felvenni a különféle feladatokhoz, de miután elkészültem velük, el is feledkeztem róluk. Minden megvolt a fejemben, amire épp szükségem volt. Ellenben kiemelném, hogy a legvégére, amikor már sorra végeztem a feladatokkal rendkívül hasznosnak bizonyult, hogy ráébrekjen olyan feladatok létrejöttére, amikről teljesen elfeledkeztem.

10.3. Szakmai ismeretek

A szakdolgozatomnak azért egy mobilalkalmazást választottam, mert a témához az illett a legjobban. Valamint mivel már rendelkeztem tapasztalattal ezen a téren, így úgy gondoltam, hogy számos tapasztalatot szerezhetek mélyebb szinten is. Elmondhatom, hogy a szakdolgozat során csak még jobban megszerettem a Java nyelvet, valamint megismerkedtem az aszinkron működéssel a gyakorlatban is.

A fejlesztésben ugyan voltak nehézségek a nagy, kiépített könyvtárak használatával,

valamint az adapterek megfelelő elkészítésével, de mostanra már úgy gondolom, hogy kényelmesen tudok mozogni ezen a téren is.

10.4. Továbbfejlesztési lehetőségek

Rengeteg ötletem van, amivel az alkalmazást tovább lehetne fejleszteni, és ami jobb felhasználói élményt biztosítana. Ezek annyira megtetszettek és a szívemhez nőttek, hogy későbbiekben szeretném megvalósítani őket.

Az egyik jelentős fejlesztés, amit eszközölnék az egy olyan modul, ami amolyan közösségi média jelleggel működne. Ugyan merésznék hangzik, de ez hosszútávú felhasználói tábor építhetne ki és új embereket is bevonozhatna. A lényege egy madaras tematikájú oldal lenne, ahol madarakhoz fűződő képeket és élményeket lehetne megosztani. Akár a saját megfigyelésinket, céljainkat is meg lehetne osztani másokkal, ezzel ösztönözve a felhasználókat a funkciók használatára.

Kicsit kevésbé jelentős elgondolás, mint a közösségi média modul, de lehetne egy olyan oldal is, ahol különböző célok lennének kitűzve akár havi szinten. Lényegében madarakat kell megfigyelni, és akár egy fotót feltölteni a megfigyelt madárról, versenyezve más felhasználókkal, hogy ki tudja hamarabb teljesíteni a cél.

Nem csak új funkciókról gondolkodtam, hanem a jelenlegi dizájn megváltoztatásán is. Mivel nem vagyok ügyes dizájnolásban, így egy kicsit több időt kellene szentelnem az újragondolására, hogy egy kényelmesebb, felhasználóbarátabb felület legyen kialakítva.

A Megfigyelések és Madárhatározó rendszereken is lehetne okosítani. A Madárhatározóba több adat megadásánál lehetne egy kis hibátűrés is a madarakat illetően. A Megfigyelések pedig kaphatna több lehetőséget és szebb kinézetet.

Gondolkodtam egy olyan modul kialakításában is, amelyikben tárképen mutatná a közelünkben lévő madárleleteket és madármegfigyelésre alkalmas területeket. Ezt gondolom valamilyen API segítségével meg lehetne oldani.

10.5. Összegzés

Úgy érzem sikerült egy használható és teljes alkalmazásnak nevezhető mobilalkalmazást készítenem. A végeredmény nekem jobban tetszik, mint amilyen módon a gondolataimban létezett. Sok érdekes technológiával és lehetőséggel ismerkedtem meg, ami a jövőben is hasznos lesz számomra, nem is beszélve arról, hogy láthattam végre, hogy mik a saját határait.

Irodalomjegyzék

TODO: irodalomjegyzék

Nyilatkozat

Alulírott Ocztos Károly Levente programtervező informatikus szakos hallgató, kijelentem, hogy a dolgozatomat a Szegedi Tudományegyetem, Informatikai Intézet Szoftverfejlesztés Tanszékén készítettem, programtervező informatikus BSc diploma megszerzése érdekében.

Kijelentem, hogy a dolgozatot más szakon korábban nem védtem meg, saját munkám eredménye, és csak a hivatkozott forrásokat (szakirodalom, eszközök, stb.) használtam fel. Tudomásul veszem, hogy szakdolgozatomat a Szegedi Tudományegyetem Diplomamunka Repozitóriumban tárolja.

2024. 12. 14. **TODO: aláírás**

Aláírás

Köszönetnyilvánítás

Szeretnék köszönetet mondani a témavezetőmnek, Dr. Bilicki Vilmos egyetemi adjunktusnak, aki vállalta a téma vezetését, útmutatást adott az ismeretlen teendőkről, felügyelte a haladásom és tanácsokkal látott el.

Szeretnék köszönetet mondani a családomnak is és barátaimnak is a támogatásukért, megértésükért, és hogy képesek voltak elviselni a megfáradt viselkedésemet, valamint nyugodt környezetet biztosítottak nekem, amikor szükségem volt rá.

Elektronikus melléklet

1. Az alkalmazás GitHub elérhetősége: <https://github.com/OcztosKaroly/Magyar-Madarak> (Utolsó megtekintés: 2024. 12. 14.)
2. A mobilalkalmazás apk-ja: **TODO** (Utolsó megtekintés: 2024. 12. 14.)